Complex Event Processing for City Officers: A Filter and Pipe Visual Approach

Complex Event Processing for City Officers: A Filter and Pipe Visual Approach

Dario Bonino and Luigi De Russis, *Member, IEEE*

***Abstract*—Administrators and operators of next generation cities will likely be required to exhibit a good understanding of technical features, data issues, and complex information that, up to few years ago, were quite far from day-to-day administration tasks. In the smart city era, the increased attention to data harvested from the city fosters a more informed approach to city administration, requiring involved operators to drive, direct, and orient technological processes in the city more effectively. Such an increasing need requires tools and platforms that can easily and effectively be controlled by non-technical people. In this paper, an approach for enabling "easier" composition of real-time data processing pipelines in smart cities is presented, exploiting a visual and block-based design approach, similar to the one adopted in the Scratch programming language for elementary school students. The proposed approach encompasses both a graphical editor and a sound methodology and workflow, to allow city operators to effectively design, develop, test, and deploy their own data processing pipelines. The editor and the workflow are described in the context of a pilot of the ALMANAC European project.**

***Index Terms*—Complex Event Processing, Block-based programming, Filter and Pipe, Big Data Analysis, Smart City, Visual Programming**

I. Introduction

Cities are playing an increasingly important role, worldwide. They are currently inhabited by nearly half the world's population [1] (68% in Europe[1]), consume 80% of the world's energy production and roughly produce 70% of the total carbon dioxide [2]. Urban-related challenges encompass, for example, (a) the adoption of information and sensing technology for better understanding the city dynamics [3]; (b) the increasing engagement of citizens in administrative and management processes, possibly leveraging the power of the crowds [4]; (c) the environmental sustainability of cities, fostering virtuous behaviors for better consuming energy and goods, and for better handling waste and pollutants [5]. In this extremely complex context, innovative city-wide ICT platforms are deemed as crucial elements for improving city management [6], [7] and, in the long term, city life quality. Several research efforts tackle the issues and challenges [8], [9] related to these "smart cities" [10], e.g., for handling huge amounts of sensors deployed in the city territory [11], [12], to manage administrative processes through open-data exchanges [13], to establish common layers of interoperability between city functions through semantic modeling [14], etc.

Among these challenges, data-handling is particularly interesting. On one hand, it requires technology capable of handling thousands of data points, continuously captured from the field. On the other hand, it requires new programming patterns and paradigms which shall be accessible to persons that were previously not exposed to hard technological aspects, e.g., city administrators. While many researchers are investigating methods and tools for Complex Event Processing (CEP) [15], [16] and Big Data [17], [18] in the context of smart cities, few approaches explore interaction patterns, programming languages and/or interfaces for enabling non-experts to define on-line data processing pipelines, as it will likely be needed in next generation cities.

In this paper we explicitly target this challenge by proposing a block-based and visual language for defining complex event processing pipelines through composition of simple processing stages, namely blocks. We exploit the proven paradigm of block-based programming, e.g., adopted by the MIT Scratch programming language [19] to enable elementary school students to approach computer science. This paradigm, integrated with the well known pipe and filter pattern [20], permits to design a language composed by simple, easy to understand, processing blocks that composed together can generate complex and articulated processing pipelines. Along with a graphical editor to define complex event processing pipelines, the paper presents a sound methodology and workflow to allow city operators to effectively design, develop, test, and deploy the pipelines they would like to define. Both the editor and the workflow presented here are described in the context of a pilot of the ALMANAC European project, in particular the pilot held in the city of Turin, in Italy.

II. Related Works

Complex Event Processing and techniques for handling high-cardinality, high-frequency data streams typical of the Big Data domain have been widely investigated in research. With the advent of Smart Cities, these topics gained an increased momentum related to the inherent challenges brought by the city scenario, from large-scale deployment to interoperability between stream formats, etc. Ganz et al. [21] present a survey of techniques and methods to process and transform raw sensor data into higher level abstractions, which are

Dario Bonino is with the Pervasive Technologies Research Area of the Istituto Superiore Mario Boella, 10138 Torino, Italy. email: dario.bonino@ismb.it

Luigi De Russis is with the Department of Control and Computer Engineering at Politecnico di Torino, 10129 Torino, Italy. email: luigi.derussis@polito.it

Manuscript received December 29, 2016; revised June 14, 2017.

[1]According to Eurostat, http://ec.europa.eu/eurostat/documents/3217494/5728777/KS-HA-11-001-EN.PDF, last visited on June 14, 2017.

human and/or machine understandable with clear references to Internet of Things and Smart Cities scenarios. Among the available approaches and techniques, a rather evident trend emerges which, on one side recognize that CEP languages are too complex for non-expert users that are required to define processing tasks, and on the other side considers semantic characterization and matching of events a viable solution to address smart city scenarios. Anicic et al. [22], define two high-level languages for specifying event patterns named "ETALIS Language for Event" and "Event Processing SPARQL" which aim at bridging the gap between domain-expert knowledge and CEP query technicalities. Their approach is similar to the one proposed in this paper as it aims at defining more "effective" and "easy-to-adopt" languages for CEP, with semantic-based definition of events. The proposed DFL moves, in fact, in the same direction and foresees the adoption of SPARQL-based identification of device tuples, in background, to speed-up the process of instantiating CEP pipelines (chains). In the work of Taylor et al. [23] ontologies are used as a basis for the definition of contextualized complex events of interest which are translated to selections and temporal combinations of streamed messages. Supported by description logic reasoning, the event descriptions defined by Taylor et al. are translated to the native language of a commercial CEP engine, and executed under the control of the CEP. This approach shares with the DFL the underlying idea of hiding the complexity of CEP under a more suitable language, exploiting semantics for achieving stream format interoperability and stream matching. However, it targets a different community of experts able to successfully understand, and master description logic specification of event processing patterns. The Data Fusion Language (DFL) exploited by the block-based approach, on the contrary, targets persons which are not skilled in CEP or Big Data analysis but in city administration and organization. As such, its main goal is to trade-off expressiveness and flexibility of CEP languages with easier composition and understandability by non-experts. Stream matching in large scale scenarios is a widely recognized issue, also shared by the DFL. Hasan et al. [24] examine the requirement of event semantic decoupling and discuss approximate semantic event matching and the consequences it implies for event processing systems. They, in particular define a semantic event matcher and evaluate the suitability of an approximate hybrid matcher based on both thesauri-based and distributional semantics-based similarity and relatedness measures. This approach might be considered for semantic stream matching required by the DFL for wild-card instantiation of chains involving multiple devices.

III. The "Bad Smell" Use Case

To demonstrate the *functionality* and test the *feasibility* of the block-based and visual approach for data processing pipelines, we present a use case named **Bad Smell**. This use case will be widely used in the remainder of the paper, especially for describing and contextualizing the proposed workflow for city officers. The realization of the Bad Smell use case will be contextualized in a pilot of the ALMANAC project (briefly described in Section IV). To realize the use case, no specialized code or external tool will be used. A secondary purpose of the use case is to demonstrate the type of processing chains that city operators may realize in a smart city context. Although this and similar use cases can be built with other methods (e.g., CEP languages), these include knowledge on specific domains and other advanced skills that many city officers do not possess.

The *Bad Smell* use case arises from a real need of an ALMANAC pilot: to know when waste bins, distributed in the entire smart city, are full and need to be emptied. This information is useful to provide a more efficient and precise waste management service. Such an information is, furthermore, particularly important if we consider *organic* waste bins: they can generate a "bad smell" even if they are not totally full. Therefore, a city officer could express the following processing need: *define a processing pipeline that generates alerts when organic waste bins in the entire city might generate unpleasant smell in the roads.*

The city officer, obviously, needs to better formalize her need in a series of conditions to be checked. For the purpose of this paper, the city officers might define the rule to generate "Bad Smell" alerts if a given waste bin is nearly full (e.g., its fill level is greater than 80% of its capacity) and the outside temperature is higher than 15°C.

IV. Background

Before diving in the details of the block-based approach and the proposed methodology and workflow, some background information could be useful to better understand the previous and related works employed in this work. In particular, the *spChains* framework, the *Data Fusion Language* adopted by the ALMANAC project, and a reference *ontology* will be introduced.

A. The ALMANAC project

The approach presented in this paper is contextualized and exemplified within the pilot held in Turin, Italy of the ALMANAC project. ALMANAC is a EU-funded project which aims at designing and developing a Smart City Platform [25] that integrates technologies and paradigms typical of the Internet of Things with edge networks, by exploiting a cloud-based, federated services approach. The project tackles the full stack of challenges involving smart city platforms, from low-level sensor interfacing and data capture, to high-level support to city processes and policies.

B. SpChains

SpChains [26] is a block-based stream processing framework, which represents monitoring (and alerting) tasks as reusable and modular "processing chains" built atop of a set of 14 standard, and extensible, stream processing blocks (Figure 1 reports a sample "smell-detection" chain).

Each block encapsulates a single (parametrized) stream query (see Figure 2), e.g., a windowed average or a threshold check, and can be cascaded to other blocks to obtain complex elaboration chains using a pipes-and-filter composition pattern [20].

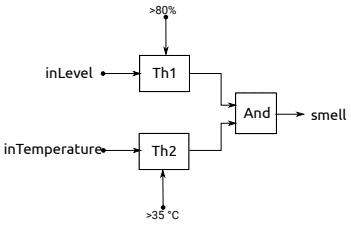

Fig. 1. Bad smell detection.

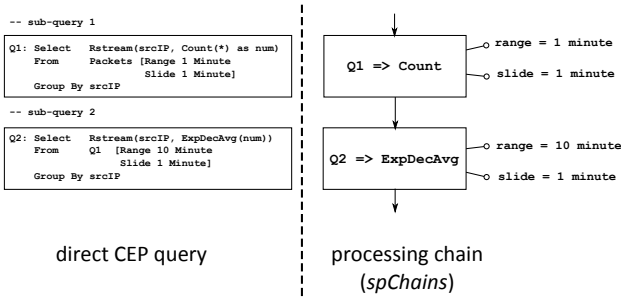

Fig. 2. Direct CEP query mapped on processing blocks.

SpChains is suitable for small deployments, with few sensors, as it requires to manually define each single processing chain (i.e., by composing JSON documents). However, when moving to smart city scenarios, where the number of sensors is huge and possibly unknown, defining chains becomes a daunting task, almost impossible to effectively accomplish. In particular, three main obstacles in adopting spChains in a smart city scenario can be identified: 1) single chain instantiation, 2) exact and explicit source matching, 3) verbosity of the chain definition language (Figure 3 shows the bad smell chain instantiation). The first obstacle requires the chain developer to fully specify every chains, even if they apply the same elaboration to different data sources. The second, requires a chain to explicitly and uniquely identify target data-sources, at design time. The third issue makes chain instantiation an error-prone, and annoying task, which might prevent adoption of the block-based stream processing paradigm.

```
{
  "chains": [ { "id": "bad_smell_1",
        "blocks" : [
          {"id":"Th1", "function":"threshold", "params":[
              {"name":"threshold", "value":"80", "uom":"%"},
              {"name":"mode", "value":"rising"}]},
          {"id":"Th2", "function":"threshold", "params":[
              {"name":"threshold", "value":"35", "uom":"Celsius"},
              {"name":"mode", "value":"rising"}]},
          {"id":"And","function":"and"}
        ],
        "connections":[
          {"from":{"blockId":"Th1", "ioId":"out"},
            "to":{"blockId":"And", "ioId":"in1"}},
          {"from":{"blockId":"Th2", "ioId":"out"},
            "to": {"blockId":"And", "ioId":"in2"}}],
        "inputs":[
          {"blockId":"Th1", "port":"in", "ioId":"inLevel"},
          {"blockId":"Th2", "port":"in", "ioId":"inTemperature"}],
        "outputs":[
          {"blockId":"And", "port":"out", "ioId":"smell"}]}],
        "bindings":[
          {"fromSources":[{"sourceId":"WasteBin36754", "dataStream":[
            {"streamId":"Temperature_acd4537fd", "ioId" : "inTemperature"},
            {"streamId":"FillLevel_cgajh74629", "ioId" : "inLevel"}]}],
           "toDrains" : [{"drainId":"bad_smell_36754", "ioId":"smell"}]}]
}
```

Fig. 3. Bad smell chain definition and binding to WasteBin36754, in the updated JSON syntax.

C. ALMANAC Data Fusion Language

To overcome the spChains issues, Bonino et al. [25] proposed the adoption of a new modeling primitive, namely the *template*, and of a new, metadata-based, template instantiation algorithm named *wild-card template binding*. The resulting block-based CEP definition language is called ALMANAC Data Fusion Language (DFL).

Template

In the ALMANAC DFL, the single chain instantiation issue is tackled, for smart city scenarios, by means of the new *template* modeling primitive. Templates are special classes of stream processing chains whose inputs and outputs are left open by exploiting syntactical place-holders, and can be later matched to different sources, and drains (Figure 4).

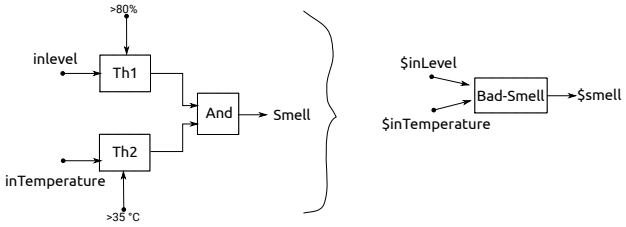

Fig. 4. The *Template* concept. On the left, the original processing chain. On the right, the corresponding template.

Templates define "prototypes" of processing chains (which can also be seen as a virtual and complex blocks) that can be instantiated in same way of any other processing block. Templates ease the processing pipeline instantiation task by reducing the amount of blocks that need to be specifically described in each elaboration chain. The more complex, and the more similar are chains to instantiate, the higher is the gain in terms of time and reduced error rate.

Wild-card template binding

The term *wild-card template binding*, introduced with the ALMANAC DFL, refers to a more flexible specification of input and output streams for the designed templates. While in the original spChains approach (shown in Figure 3), chain inputs and outputs were tightly bound to uniquely identified data streams, i.e., to specific sources and drains, in the DFL this requirement has been relaxed, and a stream-type matching mechanism is employed to permit *deployment-time* binding of different data streams to the same chain structure. Chains can be defined to be valid for classes of devices of the same kind, generating the same types of data, thus improving the overall instantiation effectiveness.

With wild card binding (whose syntax is exemplified in Figure 5), the chain specification is enriched by:

1) a sourceType "matcher", which identifies the type of device, e.g., referred to a well known domain ontology (briefly described in Section IV-D), generating the needed data streams;
2) a streamType "matcher", which identifies the type of stream "suitable" for a given chain input.

```
"bindings":[
  {
    "fromSources" : [{ "sourceType" : "smartcity:WasteBin", "dataStream" : [
      {"streamType" : "smartcity:Temperature", "ioId" : "inTemperature_genid"},
      {"streamType" : "smartcity:FillLevel", "ioId" : "inLevel_genid"}
      ]}],
    "toDrains" : [{"drainId": "bad_smell_36754", "ioId" : "smell_genid"}]
  }]
```

Fig. 5. *Wild-card template binding* syntax example, only the "bindings" section is affected. The "_genid" subfix identifies runtime-generated identifiers. Source and stream types are referred to the ALMANAC smart city ontology.

Currently, the DFL only supports the case of templates attached to a single device instance, which is reasonably straightforward. The corresponding pseudo-code is reported in Algorithm 1.

Algorithm 1 *wild-card template binding* - simple case

```
1: procedure SWC-INSTANTIATION
2:     let D be the set of "selected" devices
3:     for each device d in D do
4:         ch = template.instantiate()
5:         for each stream in device.streams do
6:             /*get the only source available*/
7:             s = getSource(ch)
8:             /*get the source stream matching the device stream */
9:             s_stream = getMatchingSourceStream(stream,s)
10:            attach(s_stream, stream)
11:        end for
12:    end for
13: end procedure
```

For templates involving multiple sources, the binding algorithm requires the definition of queries C, possibly matching tuples T of devices. Resulting device tuples are associated to the actual chain inputs by means of the "matcher" fields specified in the *wild-card template binding*. A preliminary, rough version of a suitable wild-card binding algorithm for multiple devices is still in progress but it is reported in Algorithm 2 where constraints are matched against device tuples, that can be identified, e.g., by means of SPARQL-based queries sent to the platform device catalog.

Algorithm 2 *wild-card template binding*

```
1: procedure WC-INSTANTIATION
2:     /*constraints on devices*/
3:     let C be (c_1, ..., c_i)
4:     /*tuples of devices satisfying C*/
5:     let T be (d_11, ..., d_1i), ..., (d_n1, ..., d_ni)
6:     for each tuple t in T do
7:         ch= template.instantiate()
8:         for each device d in t do
9:             /*get the source matching the current device*/
10:            s = getMatchingSource(ch,d)
11:            for each stream in d.streams do
12:                /*get the source stream matching the device stream */
13:                s_stream = getMatchingSourceStream(stream,s)
14:                attach(s_stream, stream)
15:            end for
16:        end for
17:    end for
18: end procedure
```

In summary, the process of instantiating same-structured chains reduces to a *wild-card template binding* specification integrated by a source search and match process, which can, for example, exploit native features of the smart city platform in which the language is employed.

D. The ALMANAC smart city ontology

One of the main advantages introduced by the DFL is the ability to instantiate templates by attaching data sources identified through queries, in particular SPARQL queries over a specific, yet interchangeable, domain ontology.

In ALMANAC, a smart city ontology defines the set of sensors and devices deployed in a city, and empowers effective selection of data sources for DFL templates instantiation. The ALMANAC smart city ontology[2] is built atop of the W3C SSN[3], enriched with smart city specific concepts, i.e., the notion of administrative boundaries. The overall ontology structure encompasses three layers:

1) The Smart City Layer, building on SSN and adding smart city concepts;
2) The Sensor Layer, defining specific classes of devices and sensors employed in cities;
3) The Instance Layer, hosting A-Boxes only, i.e., defining concrete individuals involved in modeled cities (e.g., in Turin).

The Smart City Layer mainly focuses on the waste and water management domain and adds contextual information on the kind of "waste bins" available in the city, the type of waste collected and the waste generation patterns and relations characterizing city quarters, administrative districts, and the municipality as a whole.

The Sensor layer defines classes (and few instances) for representing specific devices / platforms / systems corresponding to real objects deployed in a smart city (see Figure 6). In ALMANAC, such a layer includes detailed representations of both real sensors employed in the project pilots and simulated sensors adopted for scalability testing purposes. In a more generic deployment this layer will contain the representations of all sensors deployed in a smart city, in compliance with SSN.

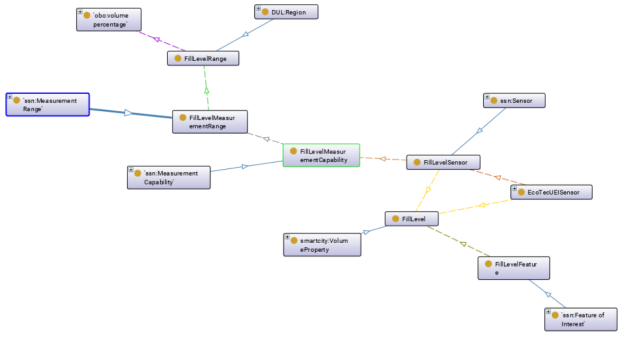

Fig. 6. Sample sensor model, in the ALMANAC smart city ontology.

Finally, the Instance layer hosts definition of individuals representing the actual devices deployed in the city. In the ALMANAC project case, for example, it contains the models of over 40k waste bins distributed on the city territory and

[2]http://www.almanac-project.eu/ontologies/smartcity.owl, last visited on November 20, 2016.

[3]https://www.w3.org/TR/vocab-ssn/, last visited on June 10, 2017.

over 20k water meter connected to houses and building in the city.

The layered structure of the ontology has two main advantages. On one side, it supports very detailed description of sensing device properties and features, accounting for correctly expressed physical quantities and measures. On the other hand, it extremely simplifies the later process of individual specification as most device features are modeled as classes and relationships.

V. Workflow Overview

To fully support the design of processing pipelines based on the ALMANAC DFL language and to address their deployment in smart cities, a clear methodology and workflow shall be defined and tailored to target users, i.e., city officers. This target user group is typically heterogeneous and might involve both people with humanistic and economics background, and persons with technical education, e.g., engineers. It is therefore safe to assume that typical users of the workflow are not expert in complex event processing and that they would be more comfortable with a possibly visual, well defined procedure to define, test, deploy, and replicate processing pipelines according to their city "administration" and "monitoring" needs.

We tackle this challenge with a full-stack approach, covering all aspects involved in the work-flow. More precisely, we define the activity of designing big-data processing pipelines for smart cities as composed by three main phases (the overall workflow organization is depicted in Figure 7):

- a *Design phase*, where the pipeline is conceived, and formulated in terms of a concatenation of simple processing blocks, as defined in spChains;
- a *Development phase* in which the pipeline is tested by attaching its input to real data sources, i.e., city sensors, and possibly refined to achieve the desired processing goals;
- a final *Deployment phase* where the prototypical pipeline developed in the former phase is wrapped as template and deployed over thousands of sensors thanks to the DFL wild-card binding algorithm, and exploiting the ALMANAC smart city ontology to select the subset of devices to be connected.

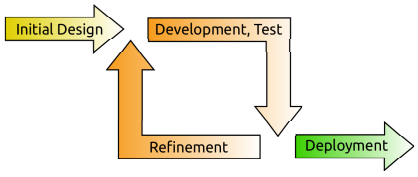

Fig. 7. The DFL overall workflow.

VI. Design Phase

In the design phase, one or more processing needs, e.g., emerging from some city-specific scenario, are formalized and decomposed in sequences of primitive blocks defined in the DFL. To perform this first step, some basic knowledge of the required operators is needed, as well as the capability to decompose a processing need into simple, consecutive operations based on the DFL blocks. To aid the city officer in the design phase, we customized NodeRED[4], a widely adopted message processing framework and visual editor, by including a new library of nodes matching the DFL basic building blocks, summarized in Table VI, and published under the CEP category in NodeRED.

TABLE I
NodeRED custom nodes implementing the DFL primitive blocks

DFL block	NodeRED library node
time-guard	node-red-contrib-time-guard
threshold	node-red-contrib-threshold
sum	node-red-contrib-sum
scale	node-red-contrib-scale
range	node-red-contrib-range
last	node-red-contrib-last
delta	node-red-contrib-delta
average	node-red-contrib-average
and	node-red-contrib-and
abs	node-red-contrib-abs
time filter	node-red-contrib-time-filter
hysteresis threshold	node-red-contrib-hysteresis-threshold

NodeRED already implements the pipes and filter paradigm and provides a huge number of modules (a.k.a., nodes) ready to be exploited by end users, through its block-based visual interface. Given the high popularity of the tool, and the proven easiness to compose complex message processing pipelines (see Figure 8, for an example), NodeRED is the ideal candidate for the DFL design and development phase.

To better understand the initial design of a DFL pipeline, let us consider the "Bad Smell" use case described in Section III. In the scenario definition, the processing need is expressed as *"define a processing pipeline that generates alerts when organic waste bins might generate unpleasant smell in the city roads"*. This has been formalized into a set of conditions to be checked, i.e., a rule to generate "Bad Smell" alerts if a given waste bin is nearly full (fill level greater than 80% of the capacity) and the outside temperature is higher than 15°C. In turn, this "rules" can be represented by connecting primitive blocks of the DFL: two threshold-crossing blocks, and one logic AND condition. The final results, using the DFL library of nodes is rendered in the NodeRED visual interface as shown in Figure 8. The design of such a pipeline took around 2-3 minutes to a computer engineer with no knowledge about NodeRED or CEP languages.

Fig. 8. The "Bad Smell" chain design in the NodeRED visual editor.

[4]https://nodered.org/, last visited on June 14, 2017.

VII. Development and Deployment

In the development phase, a "seminal" DFL chain formalized as a set of connected NodeRED nodes (a "flow" in the NodeRED jargon), is tested against trial data and finely tuned to achieve the desired behavior. Once reached a satisfying configuration, the NodeRED flow is packed as a template ("subflow", in NodeRED jargon) and deployed on the ALMANAC Data Fusion Manager (DFM) through the DFL REST APIs (see [25] for further details).

A. DFL chain testing and tuning

To support the trial and tuning operations, the DFL NodeRED library has been extended by including a configurable random data generator able to provide events with real-valued payloads and uniformly distributed within a user-configurable range. The event values might have any unit of measure, as specified by the chain designer (i.e., the city officer), and might be generated at a configurable frequency in Hz. This last feature also allows to stress test the chain under development, once the main processing blocks are connected and tuned-up. Figure 9 shows the NodeRED configuration interface for the event generator, and its connection to an input block in a DFL chain.

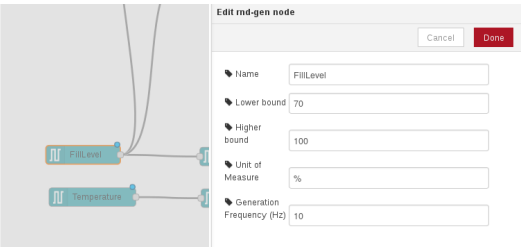

Fig. 9. Random test data generator configuration.

In every instant, the chain can be deployed locally and the outputs of its processing nodes debugged to check that they behave as expected. This test phase can also employ real-data from the smart city as input. Such an option permits to devise trial deployments where the designed processing pipeline is tested on-line for a relatively long time before being deployed in production. Figure 10, for example, reports a bad smell chain connected to two real sensors deployed in the ALMANAC pilot[5], in Turin (Italy).

It is important to emphasize that even if the development process might require a certain degree of test and trial steps, no code-digging is required as processing blocks are provided as packaged entities with tuneable parameters. Moreover, at the NodeRED level, blocks are designed to work on a parametrized payload format. In other words, since NodeRED is designed to be agnostic with respect to format of messages handled by blocks, the proposed CEP blocks comply with the same paradigm and let the user adopt whatever data format they prefer. This possibility is further enhanced by message payload manipulation primitives part of the basic NodeRED blocks. While for the deployment phase, specific data formats, e.g., the OGC Sensor Things API, are required, the NodeRED-based implementation of the development phase is data-format independent and can indeed be deployed in production, thus avoiding any concern that might arise regarding adherence to specific data formats.

Availability of new processing blocks is dependent upon the community of NodeRED developers, at the design and development phases, and on the ALMANAC community at the deployment phase. This on one hand allows exploiting the sound and proven extension mechanism of NodeRED to improve the set of supported processing primitives and to minimise the need to code new CEP operators at the end-users side. On the other hand, it allows a two pass integration mechanism were new operators are firstly introduced and shared with the wide NodeRED community and then, after a certain evaluation window, transferred to the ALMANAC platform. In this process, less used or lower quality blocks are likely to be filtered out, thus improving the overall quality of provided primitives at the smart city platform level.

B. DFL chain deployment

The final step in the DFL chain workflow involves packing the developed processing pipeline in a template and deploying the results into to the ALMANAC platform [27]. To pack the pipeline in a template, NodeRED sub-flow definition capabilities are exploited. The template operator permits, on one hand, to define a "complex" block which can be exploited in other processing pipelines, on the other hand it allows to obtain a JSON-based template description which can be easily translated into a DFL chain definition. To better understand the sub-flow packing process, let us consider the *Bad Smell* chain example. In the chain deployment step, the NodeRED flow defined in Figure 8 is wrapped into a sub-flow, e.g., named "bad-smell" (Figure 11).

Such a sub-flow is represented by the JSON description reported in Figure 12. It must be noted that without the visual editing environment of NodeRED, the process of defining such a JSON representation (or the equivalent DFL syntax) would become error prone and difficult to achieve by any human being. This gets even worse if the process shall be repeated for several chain definitions or, in case the DFL binding is not available, if it shall be applied to the huge number of, e.g., waste bins (in Turin they amount to 48000) distributed in a smart city.

Once defined the template to deploy, e.g., the bad-smell template, the deployment to the actual smart city platform, and wild-card binding to the "right" set of resources is done through a dedicated deploy procedure we designed and integrated in the NodeRED visual interface. Similarly to the normal local deployment, instantiation and binding of templates in the ALMANAC platform is done trough a dedicated "Deploy to ALMANAC" item in the top-right "Deploy" menu of the NodeRED interface (Figure 13).

Upon selecting this additional option, the city officer is offered a simple, form-based interface to select: the ALMANAC platform to which the template shall be deployed and the type

[5]http://www.almanac-project.eu/newsletters/no7_September_2016.html, last visited on December 20, 2016.

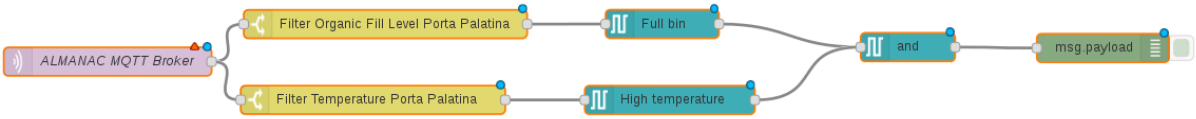

Fig. 10. Bad Smell chain test with real data from the ALMANAC Turin pilot.

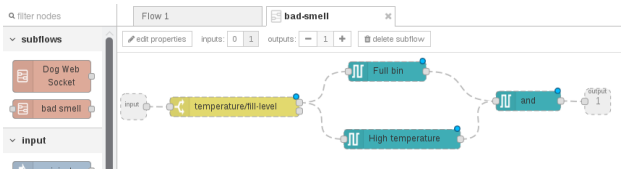

Fig. 11. The bad-smell template.


```
[{"id":"3ce6eccc.2e47b4","type":"subflow","name":"bad-smell","info":"","in":[{"x
":25,"y":83,"wires":[{"id":"975425aa.46484"}]}],"out":[{"x":742,"y":75,"
wires":[{"id":"9e08c0bc.e1ddb","port":0}]}]},
{"id":"29513301.947aec","type":"threshold","z":"3ce6eccc.2e47b4","name":"Full
bin","threshold":"80","thresholdMode":"rising","valueField":"value","x
":416.88330078125,"y":28.883331298828125,"wires":[["9e08c0bc.e1ddb"]]},{"
id":"330ed138.57cee6","type":"threshold","z":"3ce6eccc.2e47b4","name":"
High temperature","threshold":"25","thresholdMode":"rising","valueField":"
value","x":439.8833312988281,"y":132.88327026367188,"wires":[["9e08c0bc.
e1ddb"]]},
{"id":"9e08c0bc.e1ddb","type":"and","z":"3ce6eccc.2e47b4","name":"","timeWindow
":30,"timeResolution":1000,"ignoreSynch":true,"numberOfFlows":2,"
valueField":"value","x":634.88330078125,"y":74.25,"wires":[[]]},
{"id":"975425aa.46484","type":"switch","z":"3ce6eccc.2e47b4","name":"temperature
/fill-level","property":"payload.metadata.type","propertyType":"msg","
rules":[{"t":"eq","v":"http://almanac-project.eu/ontologies/smartcity.owl#
FillLevelSensor","vt":"str"},{"t":"eq","v":"http://almanac-project.eu/
ontologies/smartcity.owl#TemperatureSensor","vt":"str"}],"checkall":"true
","outputs":2,"x":181.10000610351562,"y":83.21665954589844,"wires
":[["29513301.947aec","330ed138.57cee6"],[]]},
{"id":"519ab1cf.a00c08","type":"subflow:3ce6eccc.2e47b4","z":"cba5b984.60fd4","x
":441.1000061035156,"y":151.25,"wires":[[]]}]
```


Fig. 12. The bad-smell template in the NodeRED syntax

of binding to be applied (simple vs. generalized). Depending on the latter, some additional information is required: the type of sensors (data streams) to bind in the simple case or a set of constraints (e.g., expressed in SPARQL) in the generalized version (Figure 14).

The actual deployment takes place when the "Confirm deploy" button is selected; the button causes the submission to the chosen ALMANAC DFM service (whose APIs are documented in [25]). The deployment operation may generate errors, e.g., due to wrong SPARQL syntax. Currently, these errors are simply reported back as textual error messages, however future works will tackle the issue of effectively specifying and validating constraints on possible sources for the wild-card binding algorithms.

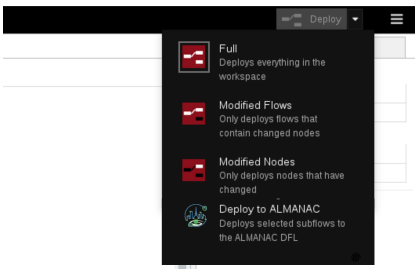

Fig. 13. Deploy to ALMANAC menu item.

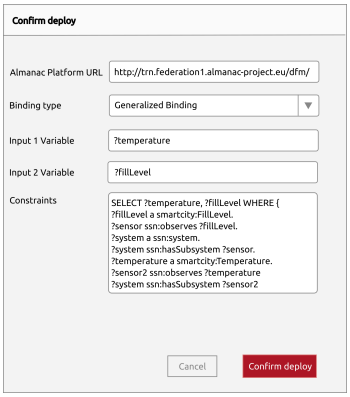

Fig. 14. Deployement on ALMANAC with Wildcard Binding.

VIII. Results

The visual language described in this paper has been adopted in the ALMANAC project as a mean to specify complex event processing chains to be executed by the ALMANAC DFL component. The platform, including the language and the deployment techniques described in previous sections, has been tested on a real-world pilot located in 2 sites in the city of Turin, Italy. Each site hosts a Underground Ecological Island (UEI) equipped with 4 waste containers[6] collecting the organic, glass, paper and not-recyclable fractions (Figure 15).

Fig. 15. One of the UEIs involved in the ALMANAC pilot, located in Via Porta Palatina

All containers were equipped with commercial-grade wireless fill-level sensors provided by EcoTec[7] and kindly funded by Amiat. Non-recyclable containers carry an RFID-based access control system which allows waste disposal from authorized citizens only. All sensors communicate with a central, legacy, data collection service also provided by EcoTec. This setting reflects quite well the typical smart city scenario, where administrative constraints and commercial aspects are strictly intertwined, often leading to deployment of heterogeneous

[6]Funded by Amiat, the Turin Waste utility, and partly supported by NordEngineering, a third party provider, see http://www.nordengineering.com/en/, last visited on June 10, 2017

[7]http://www.ecotecsolution.com/, last visited on June 10, 2017

data collection infrastructures that shall be coordinated and operated as a single complex smart city platform.

The legacy data collection system gathering data from the pilot UEIs was integrated into the ALMANAC platform thanks to a dedicated device manager, part of the platform Resource Adaptation Layer. Data collected every hour and at every waste disposal (for the non-recyclable fraction) is fed into the platform and processed through DFL primitives defined by exploiting the proposed visual language. Deployed chains involve alerting, generating dedicated alert e-mails for both the waste utility operator and the pilot technical staff, bad-smell detection, out of range waste density estimation, collection truck deviation from expected routes, etc. Alerting chains, for example, allowed the pilot to run continuously in the last 2 years, with less then 24 hours of cumulative downtime (over 690k measures collected). A dedicated data dashboard allows getting a global overview on both raw and processed data, and is currently adopted for day-to-day monitoring of the pilot site (Figure 16).

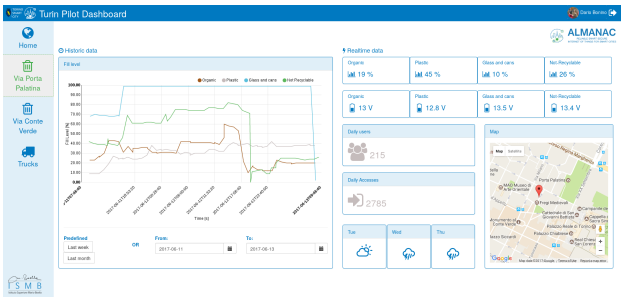

Fig. 16. The ALMANAC dashboard showing data collected on the Turin pilot.

IX. Discussion and Limitations

The presented approach is composed by a workflow for city officers and a companion block-based visual tool for creating data processing pipelines. The workflow is linear and easy to adopt for every smart city platforms like ALMANAC. The visual tool inherits from NodeRED the easiness to use and has a graphical interface that could be familiar for persons accustomed to work with IoT devices and services. The proposed workflow encompasses three phases: 1) design, 2) development (and test), 3) deployment.

The *design* phase requires more "creativity" and the capability of formalizing a data processing need in a block-based chain. This step could be challenging for some city officers, especially the ones without any technical background. Even if this issue is difficult to quantify without a proper user study with a quite wide range of city officers, the visual tool in the design phase helps to limit its possible negative effect. For this phase, in fact, the visual editor has been designed not to be linked to the smart city platform, thus it is totally robust to errors and allows novice users to freely experiment in the pipeline creation.

The next phase, *development* is more straightforward: the pipelines designed in the previous phase could be tested either with synthetic data or real data, coming from the smart city platform (in real time or from stored data). Possible problems in the designed pipeline could be edited and amended easily and without consequences. Misconfiguration of the smart city platform can be a problem for novice users of the visual editor, but they can be easily identified and reported by using synthetic, local, data.

The third and last phase of the proposed workflow could be probably the most problematic. We identified two main obstacles in the *deployment* phase, also thanks to the ALMANAC pilots:

1) the need to define additional information and constraints in SPARQL (i.e., in the form depicted in Figure 14);
2) possible errors thrown by the smart city platform (e.g., misconfiguration, errors in the pipeline definition, etc.).

Future works will tackle the issue of effectively specifying and validating constraints on possible sources for the wildcard binding algorithms, to ease the definition of additional constraints and help in solving related errors. In particular, the usage of SPARQL in the current form is critical since we are requiring city officers to possess this domain-specific knowledge. Our approach should not depend upon it.

We are aware that ontology-based modeling of sensors might generate concerns on the actual re-usability of the proposed approach, as they sometimes tend to be strictly bound to a specific knowledge domain, modeled with specific and somewhat restricting assumptions. For this reason, the entire system here presented is capable of working without ontologies and semantically annotated data, at the cost of increased deployment time[8]. A specific data format for sensory data is, however, required for deploying designed chains on the smart city platform. In the case of ALMANAC, the project consortium selected a well-known, standard OGC representation, namely the OGC Sensor Things API to limit the issues related to handling of new sensors or data.

It must be noted that in the entire workflow and, specifically in this last phase, the process of defining the needed pipeline (in the format of a JSON representation or DFL syntax) without the proposed visual editor is error prone, almost impossible to test, and difficult to achieve by any human being. This gets even worse if the process shall be repeated for several chain definitions, as expected in a smart city platform.

X. Conclusion

In a smart city context, city administrator and operators needs to be able to handle large amount of data coming from sensors distributed in the environment, and should take decisions and formulate strategies upon the analysis of such data. However, this require to exhibit a good understanding of technical features, data analysis, and complex information that are quite far from contemporary day-to-day administration tasks. The paper proposed a block-based and visual approach to tackle this problem. It presented a workflow designed for city officers and a visual tool, based on the NodeRED platform, for creating, developing, testing, and deploying data processing pipelines. The entire approach has been contextualized and

[8]Applying template binding algorithms without a uniform sensor representation is a research challenge not yet addressed.

demonstrated through a use case for waste management for an ALMANAC project pilot held in Turin, Italy. The presented approach is linear and easy to adopt for smart city platform like ALMANAC.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n.609081.

REFERENCES

[1] "State of world population 2007, unleashing the potential of urban growth," U.N. Population Fund (UNFPA), Tech. Rep. [Online]. Available: http://www.unfpa.org/public/publications/pid/408

[2] "Sustainable smart cities - building sustainable business value in changing cities," KPMG Asset Management Competence Centre, Tech. Rep., 2012.

[3] R. Petrolo, V. Loscr, and N. Mitton, "Towards a smart city based on cloud of things, a survey on the smart city vision and paradigms," *Transactions on Emerging Telecommunications Technologies*, 2015.

[4] D. Doran, S. Gokhale, and A. Dagnino, "Human sensing for smart cities," in *Advances in Social Networks Analysis and Mining (ASONAM), 2013 IEEE/ACM International Conference on*, Aug 2013, pp. 1323–1330.

[5] A. Medvedev, P. Fedchenkov, A. Zaslavsky, T. Anagnostopoulos, and S. Khoruzhnikov, *Internet of Things, Smart Spaces, and Next Generation Networks and Systems: 15th International Conference, NEW2AN 2015, and 8th Conference, ruSMART 2015, St. Petersburg, Russia, August 26-28, 2015, Proceedings*. Cham: Springer International Publishing, 2015, ch. Waste Management as an IoT-Enabled Service in Smart Cities, pp. 104–115.

[6] I. Vilajosana, J. Llosa, B. Martinez, M. Domingo-Prieto, A. Angles, and X. Vilajosana, "Bootstrapping smart cities through a self-sustainable model based on big data flows," *Communications Magazine, IEEE*, vol. 51, no. 6, p. 128134, June 2013.

[7] D. Bartlett, W. Harthoorn, J. Hogan, M. Kehoe, and R. J. Schloss, "Enabling integrated city operations," *IBM Journal of Research and Development*, vol. 55, no. 1.2, pp. 15:1–15:10, Jan 2011.

[8] A. Ojo, Z. Dzhusupova, and E. Curry, "Exploting the nature of the smart cities research landscape," *Public Administration and Information Technology*, vol. 11, 2016.

[9] M. Batty, K. Axhausen, F. Giannotti, A. Pozdnoukhov, A. Bazzani, M. Wachowicz, G. Ouzounis, and Y. Portugali, "Smart cities of the future," UCL Centre for Advanced Spatial Analysis, Tech. Rep., October 2012.

[10] "Smart cities. intelligent information and communications technology infrastructure in the government, buildings, transport, and utility domains." Pike Research, Tech. Rep., 2011. [Online]. Available: http://www.navigantresearch.com/research/smart-cities

[11] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22–32, Feb 2014.

[12] S. K. Datta and C. Bonnet, "Internet of things and m2m communications as enablers of smart city initiatives," in *Next Generation Mobile Applications, Services and Technologies, 2015 9th International Conference on*, Sept 2015, pp. 393–398.

[13] A. Cenedese, A. Zanella, L. Vangelista, and M. Zorzi, "Padova smart city: An urban internet of things experimentation," in *World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2014 IEEE 15th International Symposium on a*, June 2014, pp. 1–6.

[14] M. D'Aquin, J. Davies, and E. Motta, "Smart cities' data: Challenges and opportunities for semantic technologies," *IEEE Internet Computing*, vol. 19, no. 6, pp. 66–70, Nov-Dec 2015.

[15] D. C. Luckham, *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001.

[16] O. Etzion and P. Niblett, *Event Processing In Action*. Manning Publications and co., 2010.

[17] B. Cheng, S. Longo, F. Cirillo, M. Bauer, and E. Kovacs, "Building a big data platform for smart cities: Experience and lessons from santander," in *2015 IEEE International Congress on Big Data*, June 2015, pp. 592–599.

[18] Z. Khan, A. Anjum, and S. L. Kiani, "Cloud based big data analytics for smart future cities," in *Utility and Cloud Computing (UCC), 2013 IEEE/ACM 6th International Conference on*, Dec 2013, pp. 381–386.

[19] J. Maloney, M. Resnick, N. Rusk, B. Silverman, and E. Eastmond, "The scratch programming language and environment," *ACM Transactions on Computing Education*, vol. 10, no. 4, pp. 16:1–16:15, Nov. 2010.

[20] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern-oriented Software Architecture Volume 1*. John Wiley & Sons, 1996.

[21] F. Ganz, D. Puschmann, P. Barnaghi, and F. Carrez, "A practical evaluation of information processing and abstraction techniques for the internet of things," *IEEE Internet of Things Journal*, vol. 2, no. 4, pp. 340–354, Aug 2015.

[22] D. Anicic, S. Rudolph, P. Fodor, and N. Stojanovic, "Stream reasoning and complex event processing in etalis," *Semantic Web*, vol. 3, no. 4, pp. 397–407, 2012.

[23] K. Taylor and L. Leidinger, "Ontology-driven complex event processing in heterogeneous sensor networks," in *Proceedings of the 8th Extended Semantic Web Conference on The Semantic Web: Research and Applications - Volume Part II*, ser. ESWC'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 285–299.

[24] S. Hasan, S. O'Riain, and E. Curry, "Approximate semantic matching of heterogeneous events," in *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, ser. DEBS '12. New York, NY, USA: ACM, 2012, pp. 252–263.

[25] D. Bonino, F. Rizzo, C. Pastrone, J. A. C. Soto, M. Ahlsen, and M. Axling, "Block-based realtime big-data processing for smart cities," in *2016 IEEE International Smart Cities Conference (ISC2)*, Sept 2016, pp. 1–6.

[26] D. Bonino and F. Corno, "spChains: A Declarative Framework for Data Stream Processing in Pervasive Applications," *Procedia Computer Science*, vol. 10, pp. 316 – 323, 2012.

[27] D. Bonino, M. T. D. Alizo, A. Alapetite, T. Gilbert, M. Axling, H. Udsen, J. A. C. Soto, and M. Spirito, "Almanac: Internet of things for smart cities," in *Future Internet of Things and Cloud (FiCloud), 2015 3rd International Conference on*, Aug 2015, pp. 309–316.

Dario Bonino received his PhD in Computer Engineering from Politecnico di Torino in 2006, and his Master Degree in Electronics from Politecnico di Torino in 2002. From 2006 to 2014, he was a post-doctoral researcher at the Politecnico di Torino. He pursued research and development on Semantic Web, complex event processing applied to home and industrial automation systems, and on state chart modeling and simulation of smart environments. From September 1st, 2014, he works as researcher in the Pervasive Technologies (PerT) Research Area of the Istituto Superiore Mario Boella.

Luigi De Russis received his PhD in Computer and Control Engineering from Politecnico di Torino in 2014, and his Master Degree in Computer Engineering from Politecnico di Torino in 2010. Currently, he is a postdoc in the e-Lite research group at the Department of Control and Computer Engineering of Politecnico di Torino. His current research focuses on Human Computer Interaction, with a particular interest on interaction techniques applied to complex settings, like IoT and smart environments. He is member of IEEE and ACM.